EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

jihomoravský kraj

BEZPEČNÉ PROGRAMOVÁNÍ

Secure Code Review

Metodický list

Autor: Ing. Petr Skyva, Metodik: Ing. Miluše Jašková
Recenzent: doc. Ing. Jaroslav Dočkal, CSc.
Rok vydání: 2023

Osnova

Pracovní prostředí

Uvedení požadavků k realizaci úlohy. Např.

- Úlohu lze realizovat pouze v aplikaci DVWA
  - Damn Vulnerable Web Application
  - https://github.com/digininja/DVWA

docker ps

docker exec -ti unruffled_snyder /bin/bash

apt-get update

apt-get install nano

nano /etc/php/7.0/apache2/php.ini

Ctrl + W -> allow_url_include (On)

Ctrl + X (yes)

service apache2 restart

exit

Pro práci budeme potřebovat následující nástroje:

- Burp Suite Community
- Webový prohlížeč

Základní pojmy

TERMÍN	**VYSVĚTLENÍ**
SAST	Static Application Security Testing
DAST	Dynamic Application Security Testing
S-SDLC	Secure Software Development LifeCycle
Source	A “source” is the code that allows a vulnerability to happen
Sink	A “sink” is where the vulnerability actually happens
Black box	“Penetration” test without knowledge of the source code
White box	“Penetration” test with knowledge of the source code
Unit Tests	Unit tests are typically automated tests written and run by software developers to ensure that a section of an application (known as the "unit") meets its design and behaves as intended.
Secrets	Secrets represents information in a code, which shouldn't be there, e.g.: hardcoded credentials, API keys, …

IaC	Infrastructure as Code
PCI-DSS	Payment Card Industry Data Security Standard

1 Teoretické základy Secure Code Review

Secure Code Review má za cíl identifikovat bezpečnostní zranitelnosti v aplikaci pomocí ruční a automatizované kontroly zdrojového kódu. Člověk musí porozumět kódu aplikace, externím komponentám a konfiguraci, aby měl větší šanci případné zranitelnosti najít.

Proces auditování zdrojového kódu aplikace zároveň slouží k ověření, že jsou přítomny správné bezpečnostní a logické kontroly, že fungují tak, jak bylo zamýšleno, a že jsou na správných místech.

Secure Code Review umožňuje společnosti ujistit se, že vývojáři aplikací dodržují bezpečné vývojové techniky. Obecným pravidlem je, že penetrační test by neměl objevit žádné další (kritické) zranitelnosti v aplikaci poté, co aplikace prošla řádnou bezpečnostní kontrolou kódu. Přinejmenším by se mělo objevit jen velmi málo problémů.

Všechny kontroly bezpečnostních kódů jsou kombinací lidského úsilí a technologické podpory. Na jednom konci spektra je nezkušený člověk s textovým editorem. Na druhém konci škály je expertní bezpečnostní tým s pokročilými nástroji statické analýzy (SAST). Efektivní používání současných nástrojů zabezpečení aplikací bohužel vyžaduje poměrně seriózní úroveň odborných znalostí. Také nerozumí dynamickému toku dat nebo obchodní logice. Nástroje SAST jsou skvělé pro pokrytí a nastavení minimální základní úrovně.

K provedení Secure Code Review lze použít nástroje, které však téměř vždy vyžadují ověření nálezů člověkem. Automatizovaný nástroj nerozumí kontextu, který je základním kamenem bezpečné kontroly kódu. Nástroje typu SAST jsou skvělé pro pokrytí velkého množství kódu, nastavení minimální základní úrovně kontrol a upozornění na možné nedostatky.

1.1 Technické aspekty Secure Code Review

Secure Code Review je velmi specifický pro konkrétní aplikaci, kontrola tedy nespočívá pouze v prohledávání kódu na výskyt sady neznámých nezabezpečených modelů, ale zahrnuje také pochopení implementace kódu aplikace a logik, které jsou pro ni specifické.

Kontrolovaná aplikace mohla být navržena s některými bezpečnostními kontrolami, například s centralizovaným denylistem, validací vstupů atd. Tyto bezpečnostní kontroly je třeba pečlivě prozkoumat a zjistit, zda jsou spolehlivé pro případný specifický vektor útoku, který lze použít k jejímu obejití.

Při Secure Code Review se odhalí přesné příčiny pochybení a vysleduje se kompletní tok dat. Termín "Source to Sink analysis" znamená zjištění všech možných vstupů do aplikace (source) a způsobu jejich zpracování aplikací (sink).

Po identifikaci zranitelnosti je nutné vyhledat všechny možné případy, které se v aplikaci vyskytují. Důležité je také vyčlenění zdrojů na kontrolu, zda se tato chyba vyskytuje i v jiných produktech a aplikacích dané společnosti.

1.2 Kdy dělat Secure Code Review

Jakmile se organizace rozhodne zařadit Secure Code Review do svého interního procesu vývoje aplikací. Další důležitou otázkou, kterou je třeba si položit, je určit, v jakých fázích SDLC bude kód kontrolován.

1.2.1 Než je kód odeslán do úložiště Git (pre-commit)

Společnost zabývající se vývojem aplikací může ve svém procesu stanovit, že veškerý kód musí být zkontrolován před odesláním do úložiště zdrojového kódu (git). Nevýhodou tohoto postupu je zpomalení procesu, protože kontrola může trvat dlouho, má však mnoho výhod v tom, že se do úložiště nikdy nedostane podprůměrný kód a vedení si může být jisté, že (pokud jsou procesy dodržovány) je odeslaný kód v kvalitě, která byla stanovena.

1.2.2 Po odeslání kódu do úložiště Git (post-commit)

Zde vývojář odešle změnu kódu a poté pomocí seznamů změn v úložišti kódu odešle rozdíl ke kontrole. Výhodou tohoto postupu je, že je pro vývojáře rychlejší. Nevýhodou je, že v praxi může tato metoda vést k nižší kvalitě kódu.

Některé organizace využívající agilní metodiku, kde přidávají do svých procesů "bezpečnostní sprint". Během bezpečnostního sprintu může být kód podroben důkladnému Secure Code Review procesu s cílem odhalit co nejvíce nedostatků.

1.2.3 Během pravidelného kompletního auditu

Některé organizace mají zavedeny Secure Code Review procesy v určitých intervalech (např. jednou ročně) nebo v případě podezření, že se konkrétní zranitelnost často opakuje. Zde mohou pomoci

automatizované nástroje pro jednoduché vyhledávání řetězců v kódu (pro konkrétní vzory zranitelností) a výrazně tento proces urychlit. V tomto případě je cílem Secure Code Review celé aplikace.

1.3 Secure Code Review a dodržování právních předpisů

Mnoho společností, které jsou zodpovědné za ochranu integrity, důvěrnosti a dostupnosti (integrity, confidentiality and availability) software a dat, musí splňovat požadavky právních norem a regulací. Tento soulad je obvykle povinný, nikoli dobrovolný krol.

Právní předpisy zahrnují:

- PCI (Payment Card Industry) standards
- Regulace centrální banky
- HIPPA
- GDPR

Dodržování právních předpisů je nedílnou součástí životního cyklu vývoje software a mnoho z nich vyžaduje pravidelné Secure Code Review procesy.

2 Praktická ukázka Secure Code Review

Damn Vulnerable Web Application (DVWA) je webová aplikace vytvořená v jazyce PHP/MySQL, která je zatraceně zranitelná. Jejím hlavním cílem je pomoci profesionálům v oboru IT bezpečnosti otestovat své dovednosti a nástroje v legálním prostředí, pomoci webovým vývojářům lépe pochopit procesy zabezpečení webových aplikací a studentům a učitelům naučit se o zabezpečení webových aplikací v kontrolovaném prostředí učebny.

Cílem DVWA je procvičit některé z nejčastějších webových zranitelností s různou úrovní obtížnosti, a to pomocí jednoduchého přímočarého rozhraní. Upozorňujeme, že tento software obsahuje zdokumentované i nezdokumentované zranitelnosti. Je to záměrné. Doporučujeme vám, abyste se pokusili odhalit co nejvíce problémů.

ÚROVEŇ ZABEZPEČENÍ	VYSVĚTLENÍ
Nízká	Tato úroveň je zcela zranitelná a nemá žádné zabezpečení. Má být příkladem toho, jak se projevují zranitelnosti webových aplikací prostřednictvím špatných praktik programování.
Střední	Tato úroveň zabezpečení má sloužit především jako příklad špatné bezpečnostní praktiky, kdy se vývojář pokusil zabránit zranitelnosti, ale nepodařilo se mu zajistit dostatečnou ochranu aplikace.
Vysoká	Tato úroveň je rozšířením střední obtížnosti se směsí důkladnějších nebo alternativních (špatných) postupů při pokusu o zabezpečení kódu aplikace.
Nejvyšší	Tato úroveň by měla být zabezpečena proti všem zranitelnostem. Používá se k porovnání zranitelného zdrojového kódu se zabezpečeným zdrojovým kódem.

2.1 Command Injection

Účelem útoku typu command injection je injektovat a spustit příkazy zadané útočníkem do zranitelné aplikace. V takové situaci aplikace vykonává nežádoucí systémové příkazy a útočník ji může používat jako kterýkoli oprávněný uživatel systému. Příkazy jsou vykonávány se stejnými právy a prostředím, jaké má webová služba.

Útoky typu command injection jsou ve většině případů možné kvůli nedostatečné validaci vstupních dat, se kterými může útočník manipulovat (formuláře, cookies, hlavičky HTTP atd.). Syntaxe a příkazy se mohou v jednotlivých operačních systémech (OS), jako je Linux a Windows lišit v závislosti na požadovaných akcích.

Tento útok lze také označit jako "remote code execution" (RCE).

2.1.1 Nízká úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi “Source” a “Sink” k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```php
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
    // Get input
    $target = $_REQUEST[ 'ip' ];

    // Determine OS and execute the ping command.
    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping  ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping  -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}

?>
```

2.1.2 Střední úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi "Source" a "Sink" k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```
1. <?php
2.
3. if( isset( $_POST[ 'Submit' ]  ) ) {
4.     // Get input
5.     $target = $_REQUEST[ 'ip' ];
6.
7.     // Set blacklist
8.     $substitutions = array(
9.           '&&' => '',
10.          ';'  => '',
11.    );
12.
13.    // Remove any of the charactars in the array (blacklist).
14.    $target = str_replace( array_keys( $substitutions ),
$substitutions, $target );
15.
16.    // Determine OS and execute the ping command.
17.    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
18.          // Windows
19.          $cmd = shell_exec( 'ping  ' . $target );
20.    }
21.    else {
22.          // *nix
23.          $cmd = shell_exec( 'ping  -c 4 ' . $target );
24.    }
25.
26.    // Feedback for the end user
27.    echo "<pre>{$cmd}</pre>";
28. }
29.
30. ?>
```

2.1.3 Vysoká úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi “Source” a “Sink” k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```
1. <?php
2.
3. if( isset( $_POST[ 'Submit' ]  ) ) {
4.     // Get input
5.     $target = trim($_REQUEST[ 'ip' ]);
6.
7.     // Set blacklist
8.     $substitutions = array(
9.           '&'  => '',
10.          ';'  => '',
11.          '| ' => '',
12.          '-'  => '',
13.          '$'  => '',
14.          '('  => '',
15.          ')'  => '',
16.          '`'  => '',
17.          '||' => '',
18.    );
19.
20.    // Remove any of the charactars in the array (blacklist).
21.    $target = str_replace( array_keys( $substitutions ),
$substitutions, $target );
22.
23.    // Determine OS and execute the ping command.
24.    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
25.          // Windows
26.          $cmd = shell_exec( 'ping  ' . $target );
27.    }
28.    else {
29.          // *nix
30.          $cmd = shell_exec( 'ping  -c 4 ' . $target );
31.    }
32.
33.    // Feedback for the end user
34.    echo "<pre>{$cmd}</pre>";
35. }
36.
```

```
37. ?>
```

2.1.4 Nejvyšší úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi "Source" a "Sink" k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```
1. <?php
2.
3. if( isset( $_POST[ 'Submit' ]  ) ) {
4.     // Check Anti-CSRF token
5.     checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token'
], 'index.php' );
6.
7.     // Get input
8.     $target = $_REQUEST[ 'ip' ];
9.     $target = stripslashes( $target );
10.
11.    // Split the IP into 4 octects
12.    $octet = explode( ".", $target );
13.
14.    // Check IF each octet is an integer
15.    if( ( is_numeric( $octet[0] ) ) && ( is_numeric( $octet[1] ) ) && (
is_numeric( $octet[2] ) ) && ( is_numeric( $octet[3] ) ) && ( sizeof(
$octet ) == 4 ) ) {
16.          // If all 4 octets are int's put the IP back together.
17.          $target = $octet[0] . '.' . $octet[1] . '.' . $octet[2] . '.'
. $octet[3];
18.
19.          // Determine OS and execute the ping command.
20.          if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
21.                // Windows
22.                $cmd = shell_exec( 'ping  ' . $target );
23.          }
24.          else {
25.                // *nix
26.                $cmd = shell_exec( 'ping  -c 4 ' . $target );
27.          }
28.
29.          // Feedback for the end user
30.          echo "<pre>{$cmd}</pre>";
31.    }
32.    else {
33.          // Ops. Let the user name theres a mistake
```

```
34.         echo '<pre>ERROR: You have entered an invalid IP.</pre>';
35.   }
36. }
37.
38. // Generate Anti-CSRF token
39. generateSessionToken();
40.
41. ?>
```

2.2 File Inclusion

Některé webové aplikace umožňují uživateli nahrávat soubory na server. Později webová aplikace přistupuje k uživatelem zadanému vstupu v kontextu webové aplikace. Tímto postupem webová aplikace umožňuje potenciální spuštění škodlivého souboru, který uživatel svojí akcí uložil na server.

Pokud je zvolený soubor uložen lokálně na cílovém serveru, jedná se o tzv. (Local File Inclusion / LFI). Soubory však mohou být uloženy i na jiných vzdálených serveru, pak se jedná o útok typu (Remote File Inclusion /RFI).

2.2.1 Nízká úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi “Source” a “Sink” k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```
1. <?php
2.
3. // The page we wish to display
4. $file = $_GET[ 'page' ];
5.
6. ?>
```

2.2.2 Střední úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi “Source” a “Sink” k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```
1. <?php
2.
3. // The page we wish to display
4. $file = $_GET[ 'page' ];
5.
6. // Input validation
7. $file = str_replace( array( "http://", "https://" ), "", $file );
8. $file = str_replace( array( "../", "..\"" ), "", $file );
9.
10. ?>
```

2.2.3 Vysoká úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi "Source" a "Sink" k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```
1. <?php
2.
3. // The page we wish to display
4. $file = $_GET[ 'page' ];
5.
6. // Input validation
7. if( !fnmatch( "file*", $file ) && $file != "include.php" ) {
8.     // This isn't the page we want!
9.     echo "ERROR: File not found!";
10.    exit;
11. }
12.
13. ?>
```

2.2.4 Nejvyšší úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi “Source” a “Sink” k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```php
<?php

// The page we wish to display
$file = $_GET[ 'page' ];

// Only allow include.php or file{1..3}.php
if( $file != "include.php" && $file != "file1.php" && $file != "file2.php" && $file != "file3.php" ) {
    // This isn't the page we want!
    echo "ERROR: File not found!";
    exit;
}

?>
```

2.3 File Upload

Uploadované soubory představují pro webové aplikace významné riziko. Prvním krokem mnoha útoků je dostat do napadeného systému nějaký kód. Pak už jen útočník musí najít způsob, jak tento kód spustit. Použití funkce nahrávání souborů pomáhá útočníkovi provést první krok.

Důsledky neomezeného nahrávání souborů mohou být různé, včetně úplného ovládnutí systému, přetíženého souborového systému, útoků na backendové systémy a prostého znehodnocení dat. Záleží na tom, co aplikace s nahraným souborem dělá, včetně toho, kde je uložen.

2.3.1 Nízká úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi “Source” a “Sink” k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```
1. <?php
2.
3. if( isset( $_POST[ 'Upload' ] ) ) {
4.     // Where are we going to be writing to?
5.     $target_path  = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploads/";
6.     $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );
7.
8.     // Can we move the file to the upload folder?
9.     if( !move_uploaded_file( $_FILES[ 'uploaded' ][ 'tmp_name' ],
$target_path ) ) {
10.          // No
11.          echo '<pre>Your image was not uploaded.</pre>';
12.    }
13.    else {
14.          // Yes!
15.          echo "<pre>{$target_path} succesfully uploaded!</pre>";
16.    }
17. }
18.
19. ?>
```

2.3.2 Střední úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi "Source" a "Sink" k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```
1. <?php
2.
3. if( isset( $_POST[ 'Upload' ] ) ) {
4.     // Where are we going to be writing to?
5.     $target_path  = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploads/";
6.     $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );
7.
8.     // File information
9.     $uploaded_name = $_FILES[ 'uploaded' ][ 'name' ];
10.    $uploaded_type = $_FILES[ 'uploaded' ][ 'type' ];
11.    $uploaded_size = $_FILES[ 'uploaded' ][ 'size' ];
12.
13.    // Is it an image?
14.    if( ( $uploaded_type == "image/jpeg" || $uploaded_type ==
"image/png" ) &&
15.          ( $uploaded_size < 100000 ) ) {
16.
17.          // Can we move the file to the upload folder?
18.          if( !move_uploaded_file( $_FILES[ 'uploaded' ][ 'tmp_name' ],
$target_path ) ) {
19.                // No
20.                echo '<pre>Your image was not uploaded.</pre>';
21.          }
22.          else {
23.                // Yes!
24.                echo "<pre>{$target_path} succesfully uploaded!</pre>";
25.          }
26.    }
27.    else {
28.          // Invalid file
29.          echo '<pre>Your image was not uploaded. We can only accept
JPEG or PNG images.</pre>';
30.    }
31. }
32.
33. ?>
```

2.3.3 Vysoká úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi "Source" a "Sink" k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```
1. <?php
2.
3. if( isset( $_POST[ 'Upload' ] ) ) {
4.     // Where are we going to be writing to?
5.     $target_path  = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploads/";
6.     $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );
7.
8.     // File information
9.     $uploaded_name = $_FILES[ 'uploaded' ][ 'name' ];
10.    $uploaded_ext  = substr( $uploaded_name, strrpos( $uploaded_name,
'.' ) + 1);
11.    $uploaded_size = $_FILES[ 'uploaded' ][ 'size' ];
12.    $uploaded_tmp  = $_FILES[ 'uploaded' ][ 'tmp_name' ];
13.
14.    // Is it an image?
15.    if( ( strtolower( $uploaded_ext ) == "jpg" || strtolower(
$uploaded_ext ) == "jpeg" || strtolower( $uploaded_ext ) == "png" ) &&
16.          ( $uploaded_size < 100000 ) &&
17.          getimagesize( $uploaded_tmp ) ) {
18.
19.          // Can we move the file to the upload folder?
20.          if( !move_uploaded_file( $uploaded_tmp, $target_path ) ) {
21.                // No
22.                echo '<pre>Your image was not uploaded.</pre>';
23.          }
24.          else {
25.                // Yes!
26.                echo "<pre>{$target_path} succesfully uploaded!</pre>";
27.          }
28.    }
29.    else {
30.          // Invalid file
31.          echo '<pre>Your image was not uploaded. We can only accept
JPEG or PNG images.</pre>';
32.    }
33. }
34.
```

```
35. ?>
```

2.3.4 Nejvyšší úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi “Source” a “Sink” k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```
1. <?php
2.
3. if( isset( $_POST[ 'Upload' ] ) ) {
4.     // Check Anti-CSRF token
5.     checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token'
], 'index.php' );
6.
7.
8.     // File information
9.     $uploaded_name = $_FILES[ 'uploaded' ][ 'name' ];
10.    $uploaded_ext  = substr( $uploaded_name, strrpos( $uploaded_name,
'.' ) + 1);
11.    $uploaded_size = $_FILES[ 'uploaded' ][ 'size' ];
12.    $uploaded_type = $_FILES[ 'uploaded' ][ 'type' ];
13.    $uploaded_tmp  = $_FILES[ 'uploaded' ][ 'tmp_name' ];
14.
15.    // Where are we going to be writing to?
16.    $target_path   = DVWA_WEB_PAGE_TO_ROOT . 'hackable/uploads/';
17.    //$target_file   = basename( $uploaded_name, '.' . $uploaded_ext )
. '-';
18.    $target_file   =  md5( uniqid() . $uploaded_name ) . '.' .
$uploaded_ext;
19.    $temp_file  = ( ( ini_get( 'upload_tmp_dir' ) == '' ) ? (
sys_get_temp_dir() ) : ( ini_get( 'upload_tmp_dir' ) ) );
20.    $temp_file  .= DIRECTORY_SEPARATOR . md5( uniqid() . $uploaded_name
) . '.' . $uploaded_ext;
21.
22.    // Is it an image?
23.    if( ( strtolower( $uploaded_ext ) == 'jpg' || strtolower(
$uploaded_ext ) == 'jpeg' || strtolower( $uploaded_ext ) == 'png' ) &&
24.        ( $uploaded_size < 100000 ) &&
25.        ( $uploaded_type == 'image/jpeg' || $uploaded_type ==
'image/png' ) &&
```

```
26.         getimagesize( $uploaded_tmp ) ) {
27.
28.         // Strip any metadata, by re-encoding image (Note, using php-
Imagick is recommended over php-GD)
29.         if( $uploaded_type == 'image/jpeg' ) {
30.               $img = imagecreatefromjpeg( $uploaded_tmp );
31.               imagejpeg( $img, $temp_file, 100);
32.         }
33.         else {
34.               $img = imagecreatefrompng( $uploaded_tmp );
35.               imagepng( $img, $temp_file, 9);
36.         }
37.         imagedestroy( $img );
38.
39.         // Can we move the file to the web root from the temp folder?
40.         if( rename( $temp_file, ( getcwd() . DIRECTORY_SEPARATOR .
$target_path . $target_file ) ) ) {
41.               // Yes!
42.               echo "<pre><a
href='${target_path}${target_file}'>${target_file}</a> succesfully
uploaded!</pre>";
43.         }
44.         else {
45.               // No
46.               echo '<pre>Your image was not uploaded.</pre>';
47.         }
48.
49.         // Delete any temp files
50.         if( file_exists( $temp_file ) )
51.               unlink( $temp_file );
52.   }
53.   else {
54.         // Invalid file
55.         echo '<pre>Your image was not uploaded. We can only accept
JPEG or PNG images.</pre>';
56.   }
57. }
58.
59. // Generate Anti-CSRF token
60. generateSessionToken();
61.
62. ?>
```

2.4 Reflected Cross Site Scripting (XSS)

Útoky "Cross-Site Scripting (XSS)" jsou typem problému, při kterém jsou do jinak neškodných a důvěryhodných webových stránek vloženy škodlivé skripty (JavaScript). K útokům XSS dochází, když útočník použije webovou aplikaci k odeslání škodlivého kódu, obvykle ve formě skriptu na straně prohlížeče, jinému koncovému uživateli. Chyby, které umožňují úspěšné provedení těchto útoků, jsou poměrně rozšířené a vyskytují se všude tam, kde webová aplikace používá na výstupu vstup od uživatele, aniž by jej ověřila.

Prohlížeč koncového uživatele nemá možnost poznat, že skript je nedůvěryhodný. Protože si myslí, že skript pochází z důvěryhodného zdroje, může škodlivý skript získat přístup k souborům cookie, nebo jiným citlivým informacím uchovávaným prohlížečem.

Protože se jedná o reflected XSS, není škodlivý kód uložen ve vzdálené webové aplikaci, takže vyžaduje určité sociální inženýrství (například odkaz prostřednictvím e-mailu/chatu).

2.4.1 Nízká úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi “Source” a “Sink” k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```
1. <?php
2.
3. header ("X-XSS-Protection: 0");
4.
5. // Is there any input?
6. if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
7.     // Feedback for end user
8.     echo '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
9. }
10.
11. ?>
```

2.4.2 Střední úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi "Source" a "Sink" k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```php
<?php

header ("X-XSS-Protection: 0");

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = str_replace( '<script>', '', $_GET[ 'name' ] );

    // Feedback for end user
    echo "<pre>Hello ${name}</pre>";
}

?>
```

2.4.3 Vysoká úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi “Source” a “Sink” k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```
1. <?php
2.
3. header ("X-XSS-Protection: 0");
4.
5. // Is there any input?
6. if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
7.         // Get input
8.         $name = preg_replace( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $_GET[ 'name' ] );
9.
10.        // Feedback for end user
11.        echo "<pre>Hello ${name}</pre>";
12. }
13.
14. ?>
```

2.4.4 Nejvyšší úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi "Source" a "Sink" k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```php
<?php

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );

    // Get input
    $name = htmlspecialchars( $_GET[ 'name' ] );

    // Feedback for end user
    echo "<pre>Hello ${name}</pre>";
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

2.5 SQL Injection

Útok SQL injection spočívá ve vložení nebo dotazu SQL prostřednictvím vstupních dat z klienta do aplikace. Úspěšný útok SQL injection může číst citlivá data z databáze, měnit data databáze (vkládat/updatovat/mazat), provádět administrátorské operace nad databází (například vypnout DBMS), obnovit obsah daného souboru přítomného v souborovém systému DBMS (load_file) a v některých případech spouštět příkazy v rámci operačního systému.

Tento útok může být také nazýván "SQLi".

2.5.1 Nízká úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi “Source” a “Sink” k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```
1. <?php
2.
3. if( isset( $_REQUEST[ 'Submit' ] ) ) {
4.     // Get input
5.     $id = $_REQUEST[ 'id' ];
6.
7.     // Check database
8.     $query  = "SELECT first_name, last_name FROM users WHERE user_id =
'$id';";
9.     $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or
die( '<pre>' . ((is_object($GLOBALS["___mysqli_ston"])) ?
mysqli_error($GLOBALS["___mysqli_ston"]) : (($___mysqli_res =
mysqli_connect_error()) ? $___mysqli_res : false)) . '</pre>' );
10.
11.    // Get results
12.    while( $row = mysqli_fetch_assoc( $result ) ) {
13.          // Get values
14.          $first = $row["first_name"];
15.          $last  = $row["last_name"];
16.
17.          // Feedback for end user
18.          echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname:
{$last}</pre>";
19.    }
20.
21.    mysqli_close($GLOBALS["___mysqli_ston"]);
```

```
22. }
23.
24. ?>
```

2.5.2 Střední úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi "Source" a "Sink" k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```
1. <?php
2.
3. if( isset( $_POST[ 'Submit' ] ) ) {
4.     // Get input
5.     $id = $_POST[ 'id' ];
6.
7.     $id = mysqli_real_escape_string($GLOBALS["___mysqli_ston"], $id);
8.
9.     $query  = "SELECT first_name, last_name FROM users WHERE user_id =
$id;";
10.    $result = mysqli_query($GLOBALS["___mysqli_ston"], $query) or die(
'<pre>' . mysqli_error($GLOBALS["___mysqli_ston"]) . '</pre>' );
11.
12.    // Get results
13.    while( $row = mysqli_fetch_assoc( $result ) ) {
14.          // Display values
15.          $first = $row["first_name"];
16.          $last  = $row["last_name"];
17.
18.          // Feedback for end user
19.          echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname:
{$last}</pre>";
20.    }
21.
22. }
23.
24. // This is used later on in the index.php page
25. // Setting it here so we can close the database connection in here
like in the rest of the source scripts
26. $query  = "SELECT COUNT(*) FROM users;";
27. $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die(
'<pre>' . ((is_object($GLOBALS["___mysqli_ston"])) ?
mysqli_error($GLOBALS["___mysqli_ston"]) : (($___mysqli_res =
mysqli_connect_error()) ? $___mysqli_res : false)) . '</pre>' );
28. $number_of_rows = mysqli_fetch_row( $result )[0];
29.
30. mysqli_close($GLOBALS["___mysqli_ston"]);
```

```
31. ?>
```

2.5.3 Vysoká úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi “Source” a “Sink” k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```
1. <?php
2.
3. if( isset( $_SESSION [ 'id' ] ) ) {
4.     // Get input
5.     $id = $_SESSION[ 'id' ];
6.
7.     // Check database
8.     $query  = "SELECT first_name, last_name FROM users WHERE user_id =
'$id' LIMIT 1;";
9.     $result = mysqli_query($GLOBALS["___mysqli_ston"], $query ) or die(
'<pre>Something went wrong.</pre>' );
10.
11.    // Get results
12.    while( $row = mysqli_fetch_assoc( $result ) ) {
13.          // Get values
14.          $first = $row["first_name"];
15.          $last  = $row["last_name"];
16.
17.          // Feedback for end user
18.          echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname:
{$last}</pre>";
19.    }
20.
21.    ((is_null($___mysqli_res =
mysqli_close($GLOBALS["___mysqli_ston"]))) ? false : $___mysqli_res);

22. }
23.
24. ?>
```

2.5.4 Nejvyšší úroveň zabezpečení

- Otázky pro následující úlohu:
    1. Co vykonává uvedený zdrojový kód a v jaké aplikaci můžeme podobnout logiku najít?
    2. Na kterém řádku se nachází uživatelský vstup (Source)?
    3. Na kterém řádku se uživatelský vstup zpracovává (Sink)?
    4. Dojde mezi "Source" a "Sink" k nějaké validaci, pokud ano, je bezpečná?
    5. Dá se nějakým způsobem logika aplikaci zneužít k vyvolání bezpečnostní zranitelnosti?

```
1. <?php
2.
3. if( isset( $_GET[ 'Submit' ] ) ) {
4.     // Check Anti-CSRF token
5.     checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token'
], 'index.php' );
6.
7.     // Get input
8.     $id = $_GET[ 'id' ];
9.
10.    // Was a number entered?
11.    if(is_numeric( $id )) {
12.          // Check the database
13.          $data = $db->prepare( 'SELECT first_name, last_name FROM
users WHERE user_id = (:id) LIMIT 1;' );
14.          $data->bindParam( ':id', $id, PDO::PARAM_INT );
15.          $data->execute();
16.          $row = $data->fetch();
17.
18.          // Make sure only 1 result is returned
19.          if( $data->rowCount() == 1 ) {
20.                // Get values
21.                $first = $row[ 'first_name' ];
22.                $last  = $row[ 'last_name' ];
23.
24.                // Feedback for end user
25.                echo "<pre>ID: {$id}<br />First name: {$first}<br
/>Surname: {$last}</pre>";
26.          }
27.    }
28. }
29.
30. // Generate Anti-CSRF token
31. generateSessionToken();
32.
33. ?>
```

Shrnutí a závěr

Závěr je zhodnocením splněných cílů úlohy. Obsahuje také shrnutí nejdůležitějších poznatků v kontextu možností jejich využití.

Faktory, které je třeba zvážit při vytváření Code Review procesu

Při plánování provedení Secure Code Review je třeba vzít v úvahu více faktorů, protože každá kontrola kódu je v daném kontextu aplikace jedinečná. Je důležité vzít v úvahu všechny technické nebo obchodní faktory (jako jsou termíny a zdroje), které ovlivňují analýzu, protože tyto faktory a mohou v konečném důsledku rozhodnout o kvalitě, rychlosti a způsobu jejího provedení.

Rizika

Není možné zabezpečit vše na 100 %, proto je nezbytné stanovit priority a rizika. V ideálním světě by Secure Code Review proces měl zahrnovat veškerý kód aplikace, avšak ne každému kódu se dostane takové pozornosti, jakou by vyžadoval.

Účel a kontext

Počítačové programy mají různé účely, a proto se stupeň zabezpečení liší v závislosti na implementované funkci. Aplikace pro zprocesování online plateb bude mít vyšší bezpečnostní standardy než marketingové webové stránky. V případě platební aplikace budou mít nejvyšší prioritu údaje, jako jsou kreditní karty, avšak v případě propagačního webu je jedním z nejdůležitějších věcí, které je třeba chránit, budou přihlašovací údaje pro připojení k webovým serverům. To je další způsob, jak případnému riziku přiřadit kontext, který automatizovaný nástroj nedokáže posoudit.

Počet řádků kódu

Ukazatelem množství práce je počet řádků kódu, které je třeba zkontrolovat.

Obecně číslo řádku kódu pomáhá přesně určit místo, které je třeba opravit, a jsou velmi užitečná při kontrole oprav provedených vývojářem (například historie v úložišti kódu). Čím více řádků kódu program obsahuje, tím větší je pravděpodobnost, že se v kódu vyskytují chyby.

Programovací jazyk

Programy napsané jazycích (jako je C# nebo Java) jsou méně náchylné k určitým bezpečnostním chybám, jako je buffer overflow Při provádění Secure Code Review druh programovacího jazyka určuje typy očekávaných chyb. Je důležité znát nejčastější bezpečnostní problémy týkající se konkrétního programovacího jazyka, jehož kód má být přezkoumán, a použít tyto znalosti pro odhalení konkrétních bezpečnostních problémů.

Zdroje, načasování a konečné termíny

Jako vždy je to zásadní faktor. Kvalitní Secure Code Review složité aplikace bude trvat déle a bude vyžadovat vyšší analytické schopnosti než u jednoduchého programu. Pokud nejsou řádně zajištěny zdroje, jsou s tím spojená vyšší rizika.

Seznam použitých zdrojů

- **https://owasp.org/www-pdf-archive/OWASP_Code_Review_Guide_v2.pdf**
- **https://github.com/digininja/DVWA**